INTERPRETER, CONTROL METHOD FOR INTERPRETER, AND STORAGE MEDIUM

57)Abstract:
PROBLEM TO BE SOLVED: To execute applications at the same time without constituting all functions which operate on the interpreter as one application in advance, to actualize multi-functions, and also increase the flexibility and expansibility of program description.
SOLUTION: The interpreter 103 once receiving an indication for the execution of an application described in a programming language executes the application according to the received indication, generates storage areas for class variables needed to execute the application in a memory 201 so that they correspond to the application, and allows instructions regarding the execution of the application to access the storage areas for the class variables corresponding to the application.
TECHNICAL FIELD

[Field of the Invention] Both this inventions relate to a storage at the control approach list of the interpreter and interpreter which perform two or more applications.

PRIOR ART

[Description of the Prior Art] The interpreter conventionally equipped with two or more activation pass as an interpreter which performs interpretation of programming language and activation on a computer is offered. For example, the interpreter corresponding to a multithread is it, and if this interpreter is used, it is possible [carrying out numerical calculation] to perform network communication to coincidence.
[0003] In that case, to the thread which is each activation pass, each processing was assigned, one application program was constituted on the whole, and the interpreter was performed.

EFFECT OF THE INVENTION

[Effect of the Invention] Without constituting in advance all the

functions to make it operate by the interpreter, as one application according to this invention, as explained above, two or more applications can be performed now to coincidence, and various functions can be realized easily, and it is effective in the ability to increase the flexibility of programme description, and expandability further.
[0095] Moreover, since the instance of an interpreter can be managed with one in that case, it is effective in the ability to perform saving of memory.
[0096] Moreover, the selection executive program for choosing and performing two or more applications becomes unnecessary, and it is effective in the ability to save excessive time and effort.
[0097] Moreover, ** and the existing application also have the effectiveness of the ability to make it operate correctly that it becomes unnecessary to care about an excessive limitation by not adding a limit to the usage of a class variable in the case of programming since the activation with exact two or more applications as expected is possible, and it is easy to program.

TECHNICAL PROBLEM

[Problem(s) to be Solved by the Invention] However, the above-mentioned conventional technique had the following faults.
[0005] That is, realizing various functions, since the function which an interpreter is made to perform must be constituted as one application in advance is disagreeable ******** which is difficult and lacks in the flexibility of programme description, and expandability. Module programming which realizes one function with 1 application, or the building block system of flexibility is higher.
[0006] Therefore, the technique of performing two or more applications can be considered.
[0007] Two or more interpreters are started as the 1st technique, and there is a method of making each interpreter perform each application.
[0008] However, by this technique, two or more instances of an interpreter will exist in one computer, and it becomes a waste of memory.
[0009] Moreover, there is the approach of preparing selection executive program D for choosing and performing one piece thru/or two or more applications as the 2nd technique from two or more

applications, A, B, and C, for example, three applications.
[0010] However, if the time and effort which creates or prepares selection executive program D primarily was taken by this technique, the number of the applications to start moreover had increase and decrease, or a certain modification enters when the class of application is changed, correction of selection executive program D is needed for whenever [that], and actuation is complicated.
[0011] Moreover, by the 2nd technique of the above, it may have been said as a more fundamental problem that two or more applications were not performed correctly.
[0012] In general object oriented programming, although a class is introduced as an object type, the class variable peculiar to a class shared among two or more objects generated from the same class may be used.
[0013] By the 2nd technique of the above, the number of the instances of the class variable generated within an interpreter is one respectively. In this case, a problem will be produced if there is no unification of the volition about the usage of a class variable between applications. For example, Application B initializes with initial value 1 to the class variable y with which Class X is equipped, Application C initializes with initial value 0, and when activation is continued on the assumption that the value is not changed while each application has been initial value, a result different naturally from what was predicted will arise.
[0014] Although generating of this problem can be prevented if the arrangement thing between applications is made and a limit of direction for use is prepared about the usage of a class variable, it will strengthen a limit of application description. Moreover, the problem that a guarantee of operation cannot be offered about the existing application still remains.
[0015] While performing two or more applications to coincidence, without constituting in advance all the functions to operate this invention by the interpreter in view of the above-mentioned fault, as one application, various functions are realized and it aims at providing with a storage the control approach list of the interpreter and interpreter it was made to increase the flexibility of programme description, and expandability.

MEANS

[Means for Solving the Problem] In order to solve the above-mentioned technical problem, the interpreter concerning this invention A receipt means to be the interpreter of the programming language which supports a class variable, and to receive the directions which perform application described with said programming language, An activation means to perform said application based on the directions received by said receipt means, A generation means to match the storing field of a class variable required in the case of activation of the application by said activation means with said application, and to generate it, An access means to make the storing field of said class variable corresponding to said application access the instruction about activation of said application at the time of activation of the application by said activation means was established.

[0017] Moreover, the control approach of the interpreter concerning this invention The receipt process which is the control approach of the interpreter of the programming language which supports a class variable, and receives directions of activation of the application described with said programming language, The activation process which performs said application based on the directions received at said receipt process, The generation process which matches the storing field of a class variable required in the case of activation of the application in said activation process with said application, and generates it, The access process which makes the storing field of said class variable corresponding to said application access the instruction about activation of said application at the time of activation of the application in said activation process was established.

[0018] Moreover, the storage concerning this invention is a storage which stored the program operated as an interpreter of the programming language which supports a class variable. The receipt module for receiving the directions which perform application described with said programming language, The activation module for performing said application based on said received directions, The generation module for matching the storing field of a class variable required in the case of activation of said application with said application, and generating it, At the time of activation of said application, the program which has an access module for making the storing field of said class variable corresponding to said application access the instruction about activation of said application is stored.

[0019]

[Embodiment of the Invention] Hereafter, 1 operation gestalt of this invention is explained according to a drawing.
[0020] Drawing 1 is the block diagram of the interpreter system which is 1 operation gestalt of this invention.
[0021] In this drawing, the computer by which, as for 101, the above-mentioned interpreter system operates, the operating system (OS) with which 102 controls actuation of the whole computer 101, and 103 are interpreters which interpret and perform the cutting tool code described in object oriented programming language, and store information required for the activation on memory 201.
[0022] Moreover, the network communication module with which 104 controls the communication link with an external network, The file system module which reads 105 to the file system on a hard disk, and accesses writing, A screen output module for 106 to display a console screen or a window screen on a display, The keyboard entry module with which 107 receives the input from a keyboard, As for the hard disk which 108 stores the cutting tool code of various classes, or serves as an output destination change various entries-of-data origin, and 109, the display for a screen display and 110 are the keyboards for a key input.
[0023] Moreover, the network where 111 plays a role of a channel with an external computer, and 112a and 112b are equipped with the storage (not shown) used as an output destination change various entries-of-data origin while they store the cutting tool code of various classes, and they are the remote computer which can communicate between computers 101 about the read-out data from the above-mentioned storage, write-in data, or a user's input data.
[0024] Drawing 2 describes the contents of the above-mentioned memory 201 at the time of actuation of the above-mentioned interpreter system.
[0025] the public area 202 which is a storing field of the data shared in case the data with which the inside of memory 201 should exist only one in the above-mentioned interpreter system in this drawing, i.e., two or more applications, are performed, and the application which it is assigned for every application and is the storing field of data required for activation of each application -- it is divided into the individual field 203. three applications corresponding to them as that to which Applications A, B, and C are operating in this drawing -- the individual fields 203a, 203b, and 203c are formed.
[0026] Here, the cutting tool code of the class to which the

interpreter system loaded the time is stored in a public area 202.
[0027] A cutting tool code is the method information on the class (the instruction train of a method is included), and field information (information about an instance variable and a class variable.). Information, such as a constant data containing the initial value data of a class variable and a symbol, is expressed.
[0028] Therefore, since cutting tool codes do not differ for every application, it is stored in a public area 202. an application -- only the part of the class which overlaps rather than it stores in the individual field 203 can save memory.
[0029] In addition, each applications is not using all the classes loaded here. In case all applications are performed, a needed class is loaded here. Therefore, Application A came to have shown Class X to the public area 202 of drawing 2 , even when Application B uses Class Y and Application C is using Class Z.
[0030] on the other hand -- an application -- the data which corresponding application needs for operating correctly are stored in the individual field 203. The class variable area which needs the instance which became independent for every application, various object data areas, a stack, etc. are stored.

DESCRIPTION OF DRAWINGS

[Brief Description of the Drawings]
[Drawing 1] It is the block diagram of the interpreter system in this operation gestalt.
[Drawing 2] It is drawing having shown the memory content inside an interpreter.
[Drawing 3] It is drawing having shown the structure of a class variable table.
[Drawing 4] It is the flow chart which showed the operations sequence of an interpreter.
[Drawing 5] It is the flow chart which showed the operations sequence at the time of application activation.
[Description of Notations]
101 Computer
102 OS
103 Interpreter
104 Network Communication Module

105 File System Module
106 Screen Output Module
107 Keyboard Entry Module
108 Hard Disk
109 Display
110 Keyboard
111 Network
112 Remote Computer
201 Memory
202 Public Area
203 Application -- Individual Field
301 Class Variable Table

DETAILED DESCRIPTION

[Detailed Description of the Invention]
[0001]
[Field of the Invention] Both this inventions relate to a storage at the control approach list of the interpreter and interpreter which perform two or more applications.
[0002]
[Description of the Prior Art] The interpreter conventionally equipped with two or more activation pass as an interpreter which performs interpretation of programming language and activation on a computer is offered. For example, the interpreter corresponding to a multithread is it, and if this interpreter is used, it is possible [carrying out numerical calculation] to perform network communication to coincidence.
[0003] In that case, to the thread which is each activation pass, each processing was assigned, one application program was constituted on the whole, and the interpreter was performed.
[0004]
[Problem(s) to be Solved by the Invention] However, the above-mentioned conventional technique had the following faults.
[0005] That is, realizing various functions, since the function which an interpreter is made to perform must be constituted as one application in advance is disagreeable ******** which is difficult and lacks in the flexibility of programme description, and expandability.

Module programming which realizes one function with 1 application, or the building block system of flexibility is higher.
[0006] Therefore, the technique of performing two or more applications can be considered.
[0007] Two or more interpreters are started as the 1st technique, and there is a method of making each interpreter perform each application.
[0008] However, by this technique, two or more instances of an interpreter will exist in one computer, and it becomes a waste of memory.
[0009] Moreover, there is the approach of preparing selection executive program D for choosing and performing one piece thru/or two or more applications as the 2nd technique from two or more applications, A, B, and C, for example, three applications.
[0010] However, if the time and effort which creates or prepares selection executive program D primarily was taken by this technique, the number of the applications to start moreover had increase and decrease, or a certain modification enters when the class of application is changed, correction of selection executive program D is needed for whenever [that], and actuation is complicated.
[0011] Moreover, by the 2nd technique of the above, it may have been said as a more fundamental problem that two or more applications were not performed correctly.
[0012] In general object oriented programming, although a class is introduced as an object type, the class variable peculiar to a class shared among two or more objects generated from the same class may be used.
[0013] By the 2nd technique of the above, the number of the instances of the class variable generated within an interpreter is one respectively. In this case, a problem will be produced if there is no unification of the volition about the usage of a class variable between applications. For example, Application B initializes with initial value 1 to the class variable y with which Class X is equipped, Application C initializes with initial value 0, and when activation is continued on the assumption that the value is not changed while each application has been initial value, a result different naturally from what was predicted will arise.
[0014] Although generating of this problem can be prevented if the arrangement thing between applications is made and a limit of direction for use is prepared about the usage of a class variable, it will

strengthen a limit of application description. Moreover, the problem that a guarantee of operation cannot be offered about the existing application still remains.
[0015] While performing two or more applications to coincidence, without constituting in advance all the functions to operate this invention by the interpreter in view of the above-mentioned fault, as one application, various functions are realized and it aims at providing with a storage the control approach list of the interpreter and interpreter it was made to increase the flexibility of programme description, and expandability.
[0016]
[Means for Solving the Problem] In order to solve the above-mentioned technical problem, the interpreter concerning this invention A receipt means to be the interpreter of the programming language which supports a class variable, and to receive the directions which perform application described with said programming language, An activation means to perform said application based on the directions received by said receipt means, A generation means to match the storing field of a class variable required in the case of activation of the application by said activation means with said application, and to generate it, An access means to make the storing field of said class variable corresponding to said application access the instruction about activation of said application at the time of activation of the application by said activation means was established.
[0017] Moreover, the control approach of the interpreter concerning this invention The receipt process which is the control approach of the interpreter of the programming language which supports a class variable, and receives directions of activation of the application described with said programming language, The activation process which performs said application based on the directions received at said receipt process, The generation process which matches the storing field of a class variable required in the case of activation of the application in said activation process with said application, and generates it, The access process which makes the storing field of said class variable corresponding to said application access the instruction about activation of said application at the time of activation of the application in said activation process was established.
[0018] Moreover, the storage concerning this invention is a storage which stored the program operated as an interpreter of the

programming language which supports a class variable. The receipt module for receiving the directions which perform application described with said programming language, The activation module for performing said application based on said received directions, The generation module for matching the storing field of a class variable required in the case of activation of said application with said application, and generating it, At the time of activation of said application, the program which has an access module for making the storing field of said class variable corresponding to said application access the instruction about activation of said application is stored.
[0019]
[Embodiment of the Invention] Hereafter, 1 operation gestalt of this invention is explained according to a drawing.
[0020] Drawing 1 is the block diagram of the interpreter system which is 1 operation gestalt of this invention.
[0021] In this drawing, the computer by which, as for 101, the above-mentioned interpreter system operates, the operating system (OS) with which 102 controls actuation of the whole computer 101, and 103 are interpreters which interpret and perform the cutting tool code described in object oriented programming language, and store information required for the activation on memory 201.
[0022] Moreover, the network communication module with which 104 controls the communication link with an external network, The file system module which reads 105 to the file system on a hard disk, and accesses writing, A screen output module for 106 to display a console screen or a window screen on a display, The keyboard entry module with which 107 receives the input from a keyboard, As for the hard disk which 108 stores the cutting tool code of various classes, or serves as an output destination change various entries-of-data origin, and 109, the display for a screen display and 110 are the keyboards for a key input.
[0023] Moreover, the network where 111 plays a role of a channel with an external computer, and 112a and 112b are equipped with the storage (not shown) used as an output destination change various entries-of-data origin while they store the cutting tool code of various classes, and they are the remote computer which can communicate between computers 101 about the read-out data from the above-mentioned storage, write-in data, or a user's input data.
[0024] Drawing 2 describes the contents of the above-mentioned memory 201 at the time of actuation of the above-mentioned

interpreter system.
[0025] the public area 202 which is a storing field of the data shared in case the data with which the inside of memory 201 should exist only one in the above-mentioned interpreter system in this drawing, i.e., two or more applications, are performed, and the application which it is assigned for every application and is the storing field of data required for activation of each application -- it is divided into the individual field 203. three applications corresponding to them as that to which Applications A, B, and C are operating in this drawing -- the individual fields 203a, 203b, and 203c are formed.
[0026] Here, the cutting tool code of the class to which the interpreter system loaded the time is stored in a public area 202.
[0027] A cutting tool code is the method information on the class (the instruction train of a method is included), and field information (information about an instance variable and a class variable.). Information, such as a constant data containing the initial value data of a class variable and a symbol, is expressed.
[0028] Therefore, since cutting tool codes do not differ for every application, it is stored in a public area 202. an application -- only the part of the class which overlaps rather than it stores in the individual field 203 can save memory.
[0029] In addition, each application is not using all the classes loaded here. In case all applications are performed, a needed class is loaded here. Therefore, Application A came to have shown Class X to the public area 202 of drawing 2 , even when Application B uses Class Y and Application C is using Class Z.
[0030] on the other hand -- an application -- the data which corresponding application needs for operating correctly are stored in the individual field 203. The class variable area which needs the instance which became independent for every application, various object data areas, a stack, etc. are stored.
[0031] A class variable area should secure only the part of a required thing with the application. In drawing 2 , although Applications A, B, and C are using all the classes X, Y, and Z, when all classes are not being used, the class variable area of an unnecessary class does not need to secure.
[0032] now, the field to the class variable which the class has when application uses a class, although it is attached to one class and 1 thru/or two or more class variables can be defined and in which all writing is possible -- an application -- it is secured on the individual

field 203.
[0033] Drawing 3 is the class variable table 301 having shown it.
[0034] When it has the class variable x which can write in Class X, and y and z and Applications A and B and all C use Class X, the data storage field to 301 a-i is secured. in addition -- although the class variable area was expressed as a table as a gestalt different here from drawing 2 -- an application -- the line of A -- an application -- I think that it corresponds with the class variable area of individual field 203a.
[0035] Here, the approach is not indispensable although considered as "the class variable in which all writing is possible." If it confirms whether to be finishing [partitioning] and has not secured yet in case it is used to the class variable of each, there is also a method of reserving a partition. Since only the part of the class variable to be used should prepare a class variable area, memory usage is stopped.
[0036] Moreover, about the class variable which cannot write in the reason "whose writing was enabled", initial value is included in the cutting tool code, and since the value is eternal, it is because it is not necessary to assign a field for every application. What is necessary is just to refer to the cutting tool code of a public area 202 in quest of a value.
[0037] Actuation of an interpreter system with the above configurations is explained below using the flow chart of drawing 4 and drawing 5 .
[0038] Drawing 4 is the flow chart which showed the operations sequence of an interpreter 103.
[0039] First, an interpreter 103 specifies the application which should start as an argument, and is started.
[0040] An interpreter 103 analyzes this argument (step S401), and it is confirmed whether the application which should start in an argument exists (step S402).
[0041] When it exists, the application which should start out of it is chosen (step S403). Generally, although selection is performed in the order specified by the argument, it is not necessary to adhere to this.
[0042] Then, a new task is created through API for the multitasking which OS102 offers (application programming interface) (step S404), and an application is started on the above-mentioned task (step S405).
[0043] The reason for creating a new task is that other applications cannot be started and the purpose of the concurrency of two or more

applications is not attained until the application is completed, if a new task is not created but application is started on the task context of an interpreter 103.
[0044] After starting the 1st application as mentioned above, it investigates whether return and the application which should start remain in step S402. When first the application which should start plurality as an argument is specified, it becomes multiple times and turning about the loop formation of step S402 to the step S405.
[0045] Since the application which should start does not exist in step S402 when an interpreter 103 is started without one specifying as an argument the application which should start, or when all the specified applications finish being started, it moves to step S406.
[0046] Step S406 to the step S408 is a step which waits for the directions about starting of application or termination of an interpreter.
[0047] First, it is confirmed whether there is any assignment of the starting application by the input from a keyboard 110 (step S406).
[0048] This is made according to the following operation gestalten, for example.
[0049] That is, apart from the screen which various applications output on a display 109, the window for application starting is displayed and the prompt which stimulates the input of application which should be started in the window is displayed.
[0050] A user inputs the argument passed to the identifier of the application which should start, and application from a keyboard 110 to the prompt.
[0051] The keyboard entry module 107 detects the argument passed to the identifier of the application by which the input was carried out [above-mentioned], and which should start, and application, and notifies it to an interpreter 103.
[0052] The interpreter 103 which received the notice judges it as those of starting application with assignment at step S406, moves to step S404 and performs the step which starts the specified application. When not receiving a notice, it judges that an interpreter 103 has no assignment of starting application at step S406, and moves to step S407.
[0053] Next, at step S407, it is confirmed whether there is any assignment of the starting application by the input from a network 111.
[0054] This is made according to the following operation gestalten,

for example.
[0055] That is, the interpreter 103 opens the predetermined port beforehand for network communication. API for the network communication control which OS102 offers is used for opening. An interpreter 103 receives the information of the argument passed to the identifier of the application which should start, and application from an external instrument 112, for example, a remote computer, through the above-mentioned port using a predetermined communications protocol. In order to specify the argument passed to the identifier of the application which should start on a remote computer 112, and application, a user inputs as an example using a keyboard.
[0056] The above-mentioned information is received with the network communication module 104 as mentioned above, and it is notified to an interpreter 103.
[0057] After that, the existence of starting application is judged in step S407, the same with having mentioned above, if required, application will newly be started, otherwise, it progresses to a degree.
[0058] If it is made above, the user who is present in the location of a remote computer 112 can also operate application on a computer 101.
[0059] Finally, it is confirmed whether there is any termination instruction of an interpreter 103 (step S408).
[0060] It is confirmed whether the key input which means an interpreter termination instruction is specifically made by the same procedure as above-mentioned keyboard entry.
[0061] When there is an interpreter termination instruction, while ending all the applications that are carrying out current activation, a task is deleted, various resource release processings in which the memory which was being allocated is released are performed as a post process (step S409), and processing of an interpreter 103 is ended.
[0062] When there is no interpreter termination instruction, it returns to the check of the existence of the application which should start after step S406.
[0063] As mentioned above, the procedure shown in drawing 4 enables it to perform two or more applications to starting and coincidence by the interpreter 103.
[0064] However, that two or more applications are exact and in order to guarantee actuation as expected, a class variable area must be prepared for every application.

[0065] The procedure is explained below using the flow chart of drawing 5 .
[0066] Drawing 5 is the flow chart which showed in what kind of procedure this application would be performed by the interpreter 103 on the new task started when an interpreter 103 started application in step S405 of drawing 4 for this reason.
[0067] When application is started in step S405 as a premise, it is specified from which method of which class activation is started. Here, it will be respectively called the Maine class and the Maine method. With this operation gestalt, the above-mentioned Maine class and the Maine method are obtained from the identifier of application by the meaning.
[0068] the application first corresponding to this application -- the individual field 203 is generated on memory 201 (step S501).
[0069] Next, since the cutting tool code of the above-mentioned Maine class is loaded to memory 201, the whereabouts must be looked for. Then, the class search path which described the location which looks for the whereabouts of a class cutting tool code, and the ranking to look for is used and searched (step S502).
[0070] Class search paths are the following data.
[0071]
/dir1/aaa/bbb:/dir2/ccc:@remote1/dir3/ddd/eee:@remote2/dir4[0072] The location where this data looks for the cutting tool code of a class is directory / dir3/ddd/eee (remote1 is the identifier of remote computer 112a.) of the hard disk 108 of the (1) computer 101. /dir1/aaa/bbb Directory of the hard disk 108 of the (2) computer 101 /dir2/ccc Directory of (3) remote-computer 112a It is illustrating, and twists and storages, such as a hard disk with which a directory exists, are the directories of (4) remote-computer 112b. It is four places of /dir4 (remote2 is the identifier of remote computer 112b), and means that the sequence to look for is the order of (1) to (4). Of course, a data notation is not restricted to this.
[0073] the application of the application which corresponds that class search path as puts this class search path on the shared area 202 of memory 201, may decide to refer to that data that exists in an interpreter 103 at the time of every application activation and can also specify the class search path corresponding to that application as coincidence at the time of starting application assignment -- it puts on the individual field 203 and you may enable it to refer to different class search path for every application

[0074] Since the time and effort which searches the pass with which in the case of the latter only a required location can be specified and the class of the application does not exist for the application as class search path can be saved, the rate of class retrieval improves.
[0075] Now, if the class search path explained above is used and the whereabouts of the cutting tool code of the Maine class is discovered, this is loaded to a public area 202 (step S503).
[0076] Subsequently, the loaded cutting tool code is analyzed and it is confirmed whether there is any class variable (step S504).
[0077] the case where it is although processing about a class variable was not performed when there was nothing -- first -- a class variable area -- an application -- it generates in the individual field 203 (step S505).
[0078] When the initial value data to a class variable are described or a class variable initialization method exists in a cutting tool code, there is the need for class variable initialization (step S506).
[0079] A class variable is initialized according to each case (step S507). When there are initial value data to a class variable, specifically, a class variable area is initialized by these initial value data. Or when a class variable initialization method exists, the method is performed by the interpreter 103. That is, before performing the Maine method, a class variable initialization method is performed. The method of activation is the same as the method of activation of the parts of step S508 mentioned later - step S513.
[0080] As mentioned above, after initializing a class variable, it is PC (program counter.) to the head of the instruction train of the Maine method still more. the interpretation in the method instruction train of an interpreter 103, and an activation location -- being shown -- it sets (step S508) and goes into the method running phase.
[0081] First, the instruction of the location which PC shows is interpreted (step S509).
[0082] And if PC is moved to (step S510) and the instruction which should be executed next in the control flow of the instruction train of a method by only executing the instruction (step S511) (step S512) and application is not completed on the occasion of activation of the instruction when a new class is not required (step S513), it returns and repeats to step S509.
[0083] When the new class which is not loaded until now is needed at the time of activation of an instruction, the cutting tool code of a new class is loaded in the same procedure as the case where (step

S510) and the cutting tool code of the above-mentioned Maine class are loaded, and generation and initialization of the field of the class variable are performed (step S515 - step S520). And the instruction execution of step S511 is continued.
[0084] In the instruction execution of step S511, when it is that to which this instruction accesses a class variable, the activation is made as follows. The field of the class variable for access is prepared by the step of step S518 - step S520 (or step S505 - step S507), and is already in the accessible condition. From the information which shows the application under activation, and the information on the class variable to access, an interpreter 103 specifies the class variable area which should be accessed, and accesses there.
[0085] What is necessary is just to take out the data stored in the field of 301e, when are explained using drawing 3 and the instruction which it is going to execute during activation of Application B after this is what takes out the value of the class variable y of Class X.
[0086] Similarly, since the field of 301b will be accessed at the time of activation of Application A and a 301h field will be accessed at the time of activation of Application C, even if it has accessed the seemingly same class variable of an instruction of a method, it is accessed by different field with different application.
[0087] If it is made above, since each application can use a class variable, without caring about the usage of the class variable of other applications, there are few limits on programming and its possibility that a bug will arise also decreases.
[0088] Now, when it carries out executing a return instruction from the Maine method etc. and application is ended, the post process of releasing the resource which was being used with (step S513) and its application is performed (step S514). the application corresponding to the application -- deletion of the individual field 203 is also performed here.
[0089] Moreover, as long as there is a class which was being used only with the application, since the class is unnecessary, the unload of the cutting tool code may already be carried out from a public area 202. If it is made such, memory 201 can be used effectively.
[0090] The counter which shows how many applications are using the class may be formed to each class. In that case, the counter of the class which the application used is reduced in a post process (step S514). And when a counter is set to 0, I hear that there is not application which is already using the class, and it can carry out the

unload of the class. In this way, the class which two or more applications were using can also carry out an unload.
[0091] Moreover, although premised on the public area 202 which loaded the cutting tool code being on RAM in the above-mentioned example, you may place on ROM. A cutting tool code is not concerned with the situation of application of operation, but it is because it is eternal. In that case, the step of loading and an unload becomes unnecessary and class search path data and a retrieval step also become unnecessary further.
[0092] In the above, this invention has been explained based on the above-mentioned operation gestalt. Even if this invention uses hardware, it is realizable with software.
[0093] Therefore, if it is read and performed, the storage which memorized the program code of the software which realizes the function of the operation gestalt mentioned above can also realize an above-mentioned function, and constitutes this invention.
[0094]
[Effect of the Invention] Without constituting in advance all the functions to make it operate by the interpreter, as one application according to this invention, as explained above, two or more applications can be performed now to coincidence, and various functions can be realized easily, and it is effective in the ability to increase the flexibility of programme description, and expandability further.
[0095] Moreover, since the instance of an interpreter can be managed with one in that case, it is effective in the ability to perform saving of memory.
[0096] Moreover, the selection executive program for choosing and performing two or more applications becomes unnecessary, and it is effective in the ability to save excessive time and effort.
[0097] Moreover, ** and the existing application also have the effectiveness of the ability to make it operate correctly that it becomes unnecessary to care about an excessive limitation by not adding a limit to the usage of a class variable in the case of programming since the activation with exact two or more applications as expected is possible, and it is easy to program.

[Translation done.]

[Claim(s)]
[Claim 1] A receipt means to be the interpreter of the programming language which supports a class variable, and to receive the directions which perform application described with said programming language, An activation means to perform said application based on the directions received by said receipt means, A generation means to match the storing field of a class variable required in the case of activation of the application by said activation means with said application, and to generate it, The interpreter characterized by providing an access-control means to make the storing field of said class variable corresponding to said application access the instruction about activation of said application at the time of activation of the application by said activation means.
[Claim 2] Said activation means is an interpreter according to claim 1 characterized by generating a task and performing said application on the task concerned.
[Claim 3] Said activation means is an interpreter according to claim 1 characterized by performing two or more applications to coincidence.
[Claim 4] The receipt process which is the control approach of the interpreter of the programming language which supports a class variable, and receives directions of activation of the application described with said programming language, The activation process which performs said application based on the directions received at said receipt process, The generation process which matches the storing field of a class variable required in the case of activation of the application in said activation process with said application, and generates it, The control approach of the interpreter characterized by providing the access-control process which makes the storing field of said class variable corresponding to said application access the instruction about activation of said application at the time of activation of the application in said activation process.
[Claim 5] The control approach of the interpreter according to claim 4 characterized by generating a task and performing said application on the task concerned in said activation process.
[Claim 6] The control approach of the interpreter according to claim 4 characterized by performing two or more applications to coincidence in said activation process.
[Claim 7] It is the storage which stored the program operated as an interpreter of the programming language which supports a class

variable. The receipt module for receiving the directions which perform application described with said programming language, The activation module for performing said application based on said received directions, The generation module for matching the storing field of a class variable required in the case of activation of said application with said application, and generating it, The storage characterized by storing the program which has an access module for making the storing field of said class variable corresponding to said application access the instruction about activation of said application at the time of activation of said application.

PATENT ABSTRACTS OF JAPAN

(11)Publication number : **2000-194568**
(43)Date of publication of application : **14.07.2000**

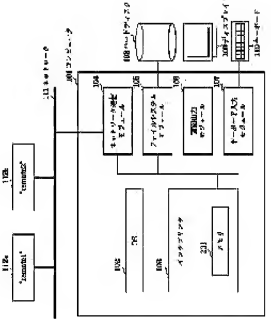
(51)Int.Cl. **G06F 9/45**
G06F 9/46

(21)Application number : **10-369920**
(22)Date of filing : **25.12.1998**

(71)Applicant : **CANON INC**
(72)Inventor : **OI KOICHI**

(54) INTERPRETER, CONTROL METHOD FOR INTERPRETER, AND STORAGE MEDIUM

(57)Abstract:
PROBLEM TO BE SOLVED: To execute applications at the same time without constituting all functions which operate on the interpreter as one application in advance, to actualize multi-functions, and also increase the flexibility and expansibility of program description.
SOLUTION: The interpreter 103 once receiving an indication for the execution of an application described in a programming language executes the application according to the received indication, generates storage areas for class variables needed to execute the application in a memory 201 so that they correspond to the application, and allows instructions regarding the execution of the application to access the storage areas for the class variables corresponding to the application.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（Ａ）**　(11)特許出願公開番号
特開2000－194568
(P2000－194568A)
(43)公開日　平成12年７月14日(2000.7.14)

(51)Int.Cl.⁷	識別記号	ＦＩ		テーマコード（参考）
Ｇ０６Ｆ　9/45		Ｃ０６Ｆ　9/44	３２０Ｃ	５Ｂ０８１
9/46	３４０	9/46	３４０Ｂ	５Ｂ０９８

審査請求　未請求　請求項の数７　ＯＬ　（全 11 頁）

(21)出願番号　特願平10－369920

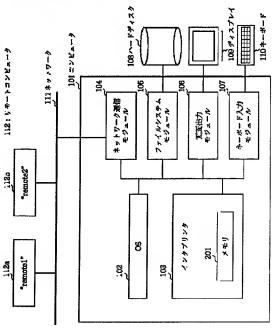
(22)出願日　平成10年12月25日(1998.12.25)

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目30番２号
(72)発明者　大井　浩一
東京都大田区下丸子３丁目30番２号キヤノン株式会社内
(74)代理人　100090538
弁理士　西山　恵三　（外２名）

Ｆターム(参考)　５Ｂ０８１　ＡＡ０９　ＤＤ０１
５Ｂ０９８　ＧＡ０４　ＧＣ０１　ＧＣ１４

(54)【発明の名称】　インタプリタ及びインタプリタの制御方法並びに記憶媒体

(57)【要約】
【課題】　インタプリタで動作させる機能を全て、事前に１個のアプリケーションとして構成することなく、複数のアプリケーションを同時に実行すると共に、多機能を実現し、プログラム記述の柔軟性、拡張性を増すようにする。
【解決手段】　インタプリタ１０３は、プログラミング言語で記述されたアプリケーションを実行する指示を受け取ると、受け取られた指示に基づいて、アプリケーションを実行し、アプリケーションの実行の際に必要なクラス変数の格納領域を、アプリケーションに対応付けてメモリ２０１内に生成し、アプリケーションの実行時に、アプリケーションの実行に関する命令をアプリケーションに対応したクラス変数の格納領域にアクセスさせる。



【特許請求の範囲】
【請求項１】　クラス変数をサポートするプログラミング言語のインタプリタであって、
前記プログラミング言語で記述されたアプリケーションを実行する指示を受け取る受取手段と、
前記受取手段により受け取られた指示に基づいて、前記アプリケーションを実行する実行手段と、
前記実行手段によるアプリケーションの実行の際に必要なクラス変数の格納領域を、前記アプリケーションに対応付けて生成する生成手段と、
前記実行手段によるアプリケーションの実行時に、前記アプリケーションの実行に関する命令を前記アプリケーションに対応した前記クラス変数の格納領域にアクセスさせるアクセス制御手段とを具備したことを特徴とするインタプリタ。
【請求項２】　前記実行手段は、タスクを生成し、当該タスク上で前記アプリケーションを実行することを特徴とする請求項１記載のインタプリタ。
【請求項３】　前記実行手段は、複数のアプリケーションを同時に実行することを特徴とする請求項１記載のインタプリタ。
【請求項４】　クラス変数をサポートするプログラミング言語のインタプリタの制御方法であって、
前記プログラミング言語で記述されたアプリケーションの実行の指示を受け取る受取工程と、
前記受取工程で受け取られた指示に基づいて、前記アプリケーションを実行する実行工程と、
前記実行工程でのアプリケーションの実行の際に必要なクラス変数の格納領域を、前記アプリケーションに対応付けて生成する生成工程と、
前記実行工程でのアプリケーションの実行時に、前記アプリケーションの実行に関する命令を前記アプリケーションに対応した前記クラス変数の格納領域にアクセスさせるアクセス制御工程とを具備したことを特徴とするインタプリタの制御方法。
【請求項５】　前記実行工程において、タスクを生成し、当該タスク上で前記アプリケーションを実行することを特徴とする請求項４記載のインタプリタの制御方法。
【請求項６】　前記実行工程において、複数のアプリケーションを同時に実行することを特徴とする請求項４記載のインタプリタの制御方法。
【請求項７】　クラス変数をサポートするプログラミング言語のインタプリタとして機能させるプログラムを格納した記憶媒体であって、
前記プログラミング言語で記述されたアプリケーションを実行する指示を受け取るための受取モジュールと、
前記受け取られた指示に基づいて、前記アプリケーションを実行するための実行モジュールと、
前記アプリケーションの実行の際に必要なクラス変数の格納領域を、前記アプリケーションに対応付けて生成するための生成モジュールと、
前記アプリケーションの実行時に、前記アプリケーションの実行に関する命令を前記アプリケーションに対応した前記クラス変数の格納領域にアクセスさせるためのアクセスモジュールとを有するプログラムを格納したことを特徴とする記憶媒体。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、複数のアプリケーションを共に実行するインタプリタ及びインタプリタの制御方法並びに記憶媒体に関するものである。
【０００２】
【従来の技術】従来より、コンピュータ上でプログラム言語の解釈、実行を行うインタプリタとして、複数の実行パスを備えるインタプリタが提供されている。例えば、マルチスレッド対応のインタプリタがそれであり、該インタプリタを用いれば、数値計算をしながら、同時にネットワーク通信を行うということが可能である。
【０００３】その場合、各実行パスであるスレッドに対して、各処理を割り当てて、全体で１個のアプリケーションプログラムを構成し、インタプリタに実行させていた。
【０００４】
【発明が解決しようとする課題】しかしながら、上記従来技術には、以下のような欠点があった。
【０００５】すなわち、インタプリタに実行させる機能を、事前に１個のアプリケーションとして構成しておかなければならないため、多機能を実現するのが困難であり、プログラム記述の柔軟性、拡張性に欠ける嫌いがあった。１機能を１アプリケーションで実現するようなモジュールプログラミングもしくはビルディングブロック方式の方が、柔軟性が高い。
【０００６】そのため、複数のアプリケーションを実行させる手法が考えられる。
【０００７】その第１の手法として、複数のインタプリタを起動し、各インタプリタに各アプリケーションを実行させる方法がある。
【０００８】しかしながら、この手法では、１台のコンピュータ内にインタプリタの複数インスタンスが存在することになり、メモリの無駄遣いになる。
【０００９】また、第２の手法として、複数のアプリケーション、例えばＡ，Ｂ，Ｃの３個のアプリケーションから１個ないし複数のアプリケーションを選択、実行させるための選択実行プログラムＤを用意する方法がある。
【００１０】しかしながら、この手法では、そもそも選択実行プログラムＤを作成または用意する手間がかかるし、その上、起動させるアプリケーションの数に増減があった、もしくは、アプリケーションの種類が変更され

た場合など、何らかの変更が入ると、その度に選択実行プログラムDの修正が必要になり、操作が煩雑である。

【0011】また、より根本的な問題として、上記第2の手法では、複数のアプリケーションが正しく実行されないといった場合が存在する。

【0012】一般的なオブジェクト指向プログラミングにおいては、オブジェクトの型としてクラスを導入するが、同一クラスから生成された複数のオブジェクト間で共有される、クラスに固有な、クラス変数が使用される場合がある。

【0013】上記第2の手法では、インタプリタ内で生成されるクラス変数のインスタンスは各々1個である。この場合、アプリケーション間にクラス変数の使用法に関する意志の統一がないと問題を生じる。例えば、クラスXが備えるクラス変数yに対して、アプリケーションBは初期値1で初期化を行い、アプリケーションCは初期値0で初期化を行い、各々のアプリケーションが初期値のまま、値は変更されていないことを前提に実行を継続すると、当然、予測されたものと異なる結果が生じることになる。

【0014】クラス変数の使用法に関し、アプリケーション間での決め事を作り、用法の制限を設ければ、この問題の発生を防止できるが、それはアプリケーション記述の制限を強化してしまうことになる。また、既存のアプリケーションに関して動作保証できないという問題は依然として残る。

【0015】上記欠点に鑑み、本発明は、インタプリタで動作させる機能を全て、事前に1個のアプリケーションとして構成することなく、複数のアプリケーションを同時に実行すると共に、多機能を実現し、プログラム記述の柔軟性、拡張性を増すようにしたインタプリタ及びインタプリタの制御方法並びに記憶媒体を提供することを目的とする。

【0016】

【課題を解決するための手段】上記課題を解決するために、本発明に係るインタプリタは、クラス変数をサポートするプログラミング言語のインタプリタであって、前記プログラミング言語で記述されたアプリケーションを実行する指示を受け取る受取手段と、前記受取手段により受け取られた指示に基づいて、前記アプリケーションを実行する実行手段と、前記実行手段によるアプリケーションの実行の際に必要なクラス変数の格納領域を、前記アプリケーションに対応付けて生成する生成手段と、前記実行手段によるアプリケーションの実行時に、前記アプリケーションの実行に関する命令を前記アプリケーションに対応した前記クラス変数の格納領域にアクセスさせるアクセス手段とを設けた。

【0017】また、本発明に係るインタプリタの制御方法は、クラス変数をサポートするプログラミング言語のインタプリタの制御方法であって、前記プログラミング言語で記述されたアプリケーションの実行の指示を受け取る受取工程と、前記受取工程で受け取られた指示に基づいて、前記アプリケーションを実行する実行工程と、前記実行工程でのアプリケーションの実行の際に必要なクラス変数の格納領域を、前記アプリケーションに対応付けて生成する生成工程と、前記実行工程でのアプリケーションの実行時に、前記アプリケーションの実行に関する命令を前記アプリケーションに対応した前記クラス変数の格納領域にアクセスさせるアクセス工程とを設けた。

【0018】また、本発明に係る記憶媒体は、クラス変数をサポートするプログラミング言語のインタプリタとして機能させるプログラムを格納した記憶媒体であって、前記プログラミング言語で記述されたアプリケーションを実行する指示を受け取るための受取モジュールと、前記受け取られた指示に基づいて、前記アプリケーションを実行するための実行モジュールと、前記アプリケーションの実行の際に必要なクラス変数の格納領域を、前記アプリケーションに対応付けて生成するための生成モジュールと、前記アプリケーションの実行時に、前記アプリケーションの実行に関する命令を前記アプリケーションに対応した前記クラス変数の格納領域にアクセスさせるためのアクセスモジュールとを有するプログラムを格納する。

【0019】

【発明の実施の形態】以下、図面に従って本発明の一実施形態を説明する。

【0020】図1は、本発明の一実施形態であるインタプリタシステムのブロック図である。

【0021】同図において、101は上記インタプリタシステムが動作するコンピュータ、102はコンピュータ101の全体の動作を制御するオペレーティングシステム（OS）、103はオブジェクト指向プログラミング言語で記述されたバイトコードを解釈、実行するインタプリタであって、その実行に必要な情報をメモリ201上に格納している。

【0022】また、104は外部ネットワークとの通信を制御するネットワーク通信モジュール、105はハードディスク上のファイルシステムに対して読み出し、書き込みのアクセスを行うファイルシステムモジュール、106はコンソール画面、もしくはウィンドウ画面をディスプレイに表示するための画面出力モジュール、107はキーボードからの入力を受け取るキーボード入力モジュール、108は様々なクラスのバイトコードを格納し、あるいは様々なデータの入力元、出力先となるハードディスク、109は画面表示のためのディスプレイ、110はキー入力のためのキーボードである。

【0023】また、111は外部コンピュータとの通信路の役割を担うネットワーク、112a及び112bは、様々なクラスのバイトコードを格納すると共に様々

なデータの入力元、出力先となる記憶媒体（図示せず）を備え、上記記憶媒体からの読み出しデータ、書き込みデータ、あるいはユーザの入力データをコンピュータ１０１との間で通信可能なリモートコンピュータである。

【００２４】図２は、上記インタプリタシステムの動作時の上記メモリ２０１の内容を記述したものである。

【００２５】同図において、メモリ２０１の中は、上記インタプリタシステムの中で、１つだけ存在すればよいデータ、すなわち、複数のアプリケーションを実行する際に共用されるデータの格納領域である共用領域２０２と、各アプリケーション毎に割り当てられ、各アプリケーションの実行に必要なデータの格納領域であるアプリ個別領域２０３に分かれている。同図では、アプリケーションＡ，Ｂ，Ｃが動作しているものとして、それらに対応する３つのアプリ個別領域２０３ａ，２０３ｂ，２０３ｃが設けられている。

【００２６】ここで、共用領域２０２には、その時点までインタプリタシステムがロードしたクラスのバイトコードが格納される。

【００２７】バイトコードとは、そのクラスの、メソッド情報（メソッドの命令列を含む）、フィールド情報（インスタンス変数及びクラス変数に関する情報。クラス変数の初期値データを含む）、定数データ、及びシンボル等の情報を表現したものである。

【００２８】従って、バイトコードはアプリケーション毎に異なるものではないので、共用領域２０２に格納される。アプリ個別領域２０３に格納するよりも重複するクラスの分だけメモリが節約できる。

【００２９】なお、各々のアプリケーションが、ここにロードされた全てのクラスを使用しているとは限らない。全てのアプリケーションを実行していく際、必要になったクラスがここにロードされる。従って、アプリケーションＡがクラスＸを、アプリケーションＢがクラスＹを、アプリケーションＣがクラスＺを使用している場合でも、図２の共用領域２０２に示したようになる。

【００３０】一方、アプリ個別領域２０３には、対応するアプリケーションが正確に動作するのに必要なデータが格納される。アプリケーション毎に独立したインスタンスが必要な、クラス変数領域、各種オブジェクトデータ領域、スタック等が格納される。

【００３１】クラス変数領域はそのアプリケーションで必要なものの分だけ確保すればよい。図２においては、アプリケーションＡ，Ｂ，Ｃ共、クラスＸ，Ｙ，Ｚの全てを使用しているとしたが、全てのクラスを使用していない場合は、不必要なクラスのクラス変数領域は確保する必要はない。

【００３２】さて、１つのクラスに付き、１ないし複数のクラス変数を定義できるが、アプリケーションがクラスを使用する場合、そのクラスが持つ全ての書き込み可能なクラス変数に対する領域がアプリ個別領域２０３上に確保される。

【００３３】図３は、それを示したクラス変数テーブル３０１である。

【００３４】クラスＸが書き込み可能なクラス変数ｘ，ｙ，ｚを備え、アプリケーションＡ，Ｂ，Ｃ全てがクラスＸを使用する場合、３０１ａ～ｉまでのデータ格納領域が確保される。なお、ここでは図２とは異なる形態として、クラス変数領域をテーブルとして表現したが、アプリＡの行がアプリ個別領域２０３ａのクラス変数領域と対応しているものと考える。

【００３５】ここでは、「全ての書き込み可能なクラス変数」としたが、その方法が必須というわけではない。１つ１つのクラス変数に対し、それを使用する際に、領域確保済みかどうかをチェックし、まだ確保していなければ、領域確保するという方法もある。使用するクラス変数の分だけ、クラス変数領域を用意すればよいので、メモリ使用量は抑えられる。

【００３６】また、「書き込み可能」とした理由は、書き込み不可能なクラス変数については、初期値がバイトコードに含まれており、その値は不変なのであるから、アプリケーション毎に領域を割り当てる必要はないからである。値を求めるには共用領域２０２のバイトコードを参照すればよい。

【００３７】以上のような構成を持つインタプリタシステムの動作について、図４及び図５のフローチャートを用いて、以下に説明する。

【００３８】図４は、インタプリタ１０３の動作手順を示したフローチャートである。

【００３９】まず、インタプリタ１０３は、引数として起動すべきアプリケーションを指定して起動される。

【００４０】インタプリタ１０３はこの引数を解析し（ステップＳ４０１）、引数の中に起動すべきアプリケーションが存在するかチェックする（ステップＳ４０２）。

【００４１】存在する場合は、その中から起動すべきアプリケーションを選択する（ステップＳ４０３）。一般的には、選択は引数で指定された順に行うが、これにこだわらなくてもよい。

【００４２】その後、新たなタスクを、ＯＳ１０２が提供するマルチタスクのためのＡＰＩ（アプリケーションプログラミングインタフェース）を通じて作成し（ステップＳ４０４）、上記タスク上でアプリを起動する（ステップＳ４０５）。

【００４３】新たなタスクを作成する理由は、もし新たなタスクを作成せず、インタプリタ１０３のタスクコンテキスト上でアプリケーションを起動してしまっては、そのアプリケーションが終了するまで、他のアプリケーションを起動することができず、複数のアプリケーションの同時実行という目的が達成されないからである。

【００４４】以上のようにして、１つ目のアプリケーシ

ョンを起動した後は、ステップS402に戻り、起動すべきアプリケーションが残っているかを調べる。最初に、引数として複数の起動すべきアプリケーションを指定した場合は、ステップS402からステップS405のループを複数回、回ることになる。

【0045】引数として、起動すべきアプリケーションを1つも指定せずに、インタプリタ103を起動した場合、もしくは、指定されたアプリケーションを全て起動し終わった場合は、ステップS402において、起動すべきアプリケーションが存在しないので、ステップS406に移る。

【0046】ステップS406からステップS408は、アプリケーションの起動、あるいはインタプリタの終了に関する指示を待つステップである。

【0047】まず、キーボード110からの入力による起動アプリケーションの指定が有るかどうかをチェックする（ステップS406）。

【0048】これは例えば、以下のような実施形態によってなされる。

【0049】すなわち、ディスプレイ109上に、各種アプリケーションが出力する画面とは別に、アプリケーション起動のためのウィンドウを表示しておき、そのウィンドウに起動すべきアプリケーションの入力を促すプロンプトを表示しておく。

【0050】そのプロンプトに対して、ユーザが、起動すべきアプリケーションの名前、アプリケーションに渡す引数をキーボード110から入力する。

【0051】キーボード入力モジュール107は上記入力された起動すべきアプリケーションの名前、アプリケーションに渡す引数を検出し、インタプリタ103に通知する。

【0052】通知を受けたインタプリタ103はステップS406で起動アプリケーションの指定有りと判断し、ステップS404に移動して、指定されたアプリケーションを起動するステップを実行する。通知を受けない場合は、インタプリタ103はステップS406で起動アプリケーションの指定無しと判断し、ステップS407に移動する。

【0053】次に、ステップS407ではネットワーク111からの入力による起動アプリケーションの指定が有るかどうかをチェックする。

【0054】これは例えば、以下のような実施形態によってなされる。

【0055】すなわち、インタプリタ103は予め、ネットワーク通信のために所定のポートをオープンしておく。オープンにはOS102が提供するネットワーク通信制御のためのAPIを使用する。インタプリタ103は上記のポートを通じ、所定の通信プロトコルを用いて、外部機器、例えばリモートコンピュータ112から、起動すべきアプリケーションの名前、アプリケーションに渡す引数といった情報を受け取る。リモートコンピュータ112上で起動すべきアプリケーションの名前、アプリケーションに渡す引数を指定するためには、一例として、ユーザがキーボードを利用して入力する。

【0056】以上のようにして上記情報は、ネットワーク通信モジュール104によって受け取られ、インタプリタ103に通知される。

【0057】その後は、上述したのと同様に、起動アプリケーションの有り無しがステップS407において判断され、必要であればアプリケーションを新たに起動し、そうでなければ次に進む。

【0058】以上のようにすれば、リモートコンピュータ112の場所にいるユーザがコンピュータ101上でアプリケーションを動作させることもできる。

【0059】最後に、インタプリタ103の終了命令が有るかどうかをチェックする（ステップS408）。

【0060】具体的には、上述のキーボード入力と同様の手順により、インタプリタ終了命令を意味するキー入力がなされていないかをチェックする。

【0061】インタプリタ終了命令がある場合は、現在実行しているアプリケーションを全て終了すると共に、タスクを削除し、割り当てていたメモリを解放するといった、様々な資源解放処理を、終了処理として行い（ステップS409）、インタプリタ103の処理を終了する。

【0062】インタプリタ終了命令がない場合は、ステップS406以降の起動すべきアプリケーションの有り無しのチェックに戻る。

【0063】以上、図4に示す手順によって、インタプリタ103によって複数のアプリケーションを起動、同時に実行することが可能になる。

【0064】しかし、複数のアプリケーションの正確かつ期待したとおりの動作を保証するためには、クラス変数領域をアプリケーション毎に準備しなければならない。

【0065】その手順を図5のフローチャートを用いて、以下に説明する。

【0066】図5は、図4のステップS405においてインタプリタ103がアプリケーションを起動した時に、このために起動された新たなタスク上で、該アプリケーションがどのような手順でインタプリタ103によって実行されるかを示したフローチャートである。

【0067】前提として、ステップS405においてアプリケーションが起動される時、どのクラスのどのメソッドから実行が開始されるかが指定される。ここでは各々、メインクラス、メインメソッドと呼ぶことにする。本実施形態では、アプリケーションの名前から上記メインクラスとメインメソッドが一意に得られるようになっている。

【0068】まず最初に、該アプリケーションに対応す

るアプリ個別領域２０３がメモリ２０１上に生成される（ステップＳ５０１）。

【００６９】次に、上記メインクラスのバイトコードをメモリ２０１にロードするために、その在処を探さなければならない。そこで、クラスバイトコードの在処を探す場所と探す順位を記述したクラス検索パスを使用し、検索する（ステップＳ５０２）。

【００７０】クラス検索パスは次のようなデータである。

【００７１】/dir1/aaa/bbb:/dir2/ccc:@remote1/dir3/ddd/eee:@remote2/dir4

【００７２】このデータは、クラスのバイトコードを探す場所は（１）コンピュータ１０１のハードディスク１０８のディレクトリ /dir1/aaa/bbb （２）コンピュータ１０１のハードディスク１０８のディレクトリ /dir2/ccc （３）リモートコンピュータ１１２ａのディレクトリ /dir3/ddd/eee （remote1はリモートコンピュータ１１２ａの名前。ディレクトリが存在するハードディスク等の記憶媒体は図示していない）（４）リモートコンピュータ１１２ｂのディレクトリ /dir4 （remote2はリモートコンピュータ１１２ｂの名前）の４個所であり、探す順序は（１）から（４）の順であることを意味する。もちろん、データ表記はこれに限るものではない。

【００７３】このクラス検索パスは、メモリ２０１の共有領域２０２に置き、どのアプリケーション実行時にも、インタプリタ１０３中に唯一存在するそのデータを参照することにしてもよいし、起動アプリケーション指定時に、そのアプリケーションに対応するクラス検索パスも同時に指定できるようにして、そのクラス検索パスを、対応するアプリケーションのアプリ個別領域２０３に置き、アプリケーション毎に異なるクラス検索パスを参照できるようにしてもよい。

【００７４】後者の場合、クラス検索パスとして、そのアプリケーションにとって必要な場所のみを指定でき、そのアプリケーションのクラスが存在しないパスを検索する手間が省けるので、クラス検索の速度が向上する。

【００７５】さて、以上に説明したクラス検索パスを利用し、メインクラスのバイトコードの在処を探し当てると、これを共用領域２０２にロードする（ステップＳ５０３）。

【００７６】次いで、ロードしたバイトコードを解析し、クラス変数が有るかチェックする（ステップＳ５０４）。

【００７７】無かった場合はクラス変数に関する処理は行わないが、有った場合はまずクラス変数領域をアプリ個別領域２０３内に生成する（ステップＳ５０５）。

【００７８】バイトコード内に、クラス変数に対する初期値データが記述されていたり、クラス変数初期化メソッドが存在する場合は、クラス変数初期化の必要性がある（ステップＳ５０６）。

【００７９】各々の場合に応じて、クラス変数を初期化する（ステップＳ５０７）。具体的には、クラス変数に対する初期値データが有る場合は、クラス変数領域を該初期値データで初期化する。あるいは、クラス変数初期化メソッドが存在する場合は、そのメソッドをインタプリタ１０３で実行する。つまり、メインメソッドを実行する前に、クラス変数初期化メソッドを実行する。実行の仕方は、後述するステップＳ５０８～ステップＳ５１３の部分の実行の仕方と同じである。

【００８０】以上のようにして、クラス変数を初期化後、いよいよ、メインメソッドの命令列の先頭にＰＣ（プログラムカウンタ。インタプリタ１０３のメソッド命令列における解釈、実行位置を示す）をセットし（ステップＳ５０８）、メソッド実行段階に入っていく。

【００８１】まず、ＰＣが示す位置の命令を解釈する（ステップＳ５０９）。

【００８２】そして、その命令の実行に際し、新たなクラスが必要でない場合は（ステップＳ５１０）、単にその命令を実行し（ステップＳ５１１）、メソッドの命令列の制御の流れにおいて、次に実行すべき命令にＰＣを移動し（ステップＳ５１２）、アプリケーションが終了しなければ（ステップＳ５１３）、ステップＳ５０９に戻って繰り返す。

【００８３】命令の実行時、今までロードしていない、新たなクラスが必要になった場合は（ステップＳ５１０）、上述のメインクラスのバイトコードをロードした場合と同様な手順で新クラスのバイトコードをロードし、そのクラス変数の領域の生成及び初期化を行う（ステップＳ５１５～ステップＳ５２０）。そして、ステップＳ５１１の命令実行を続ける。

【００８４】ステップＳ５１１の命令実行において、該命令がクラス変数をアクセスするものであった場合、その実行は以下のようになされる。アクセス対象のクラス変数の領域は既に、ステップＳ５１８～ステップＳ５２０（あるいはステップＳ５０５～ステップＳ５０７）のステップによって用意され、アクセス可能な状態になっている。インタプリタ１０３は実行中のアプリケーションを示す情報と、アクセスするクラス変数の情報から、アクセスすべきクラス変数領域を特定し、そこにアクセスする。

【００８５】図３を用いて説明すると、アプリケーションＢの実行中、これから実行しようとしている命令がクラスＸのクラス変数ｙの値を取り出すものであった場合、３０１ｅの領域に格納されたデータを取り出せばよい。

【００８６】同様に、アプリケーションＡの実行時には、３０１ｂの領域に、アプリケーションＣの実行時には、３０１ｈの領域にアクセスすることになるから、メソッドの命令の見かけ上は同じクラス変数にアクセスしていても、異なるアプリケーションでは異なる領域にア

クセスされる。

【0087】以上のようにすれば、各アプリケーションは他のアプリケーションのクラス変数の使用法を気にすること無く、クラス変数を使用できるので、プログラミング上の制限は少なく、バグが生じる可能性も少なくなる。

【0088】さて、メインメソッドからリターン命令を実行する等して、アプリケーションを終了した場合は（ステップS513）、そのアプリケーションで使用していた資源を解放する等の終了処理を行う（ステップS514）。そのアプリケーションに対応するアプリ個別領域203の削除もここで行われる。

【0089】また、そのアプリケーションのみで使用していたクラスが有れば、もうそのクラスは不要であるので、そのバイトコードを共用領域202からアンロードしてもよい。そのようにすれば、メモリ201を有効に使用できる。

【0090】各クラスに対し、いくつのアプリケーションがそのクラスを使用しているかを示すカウンタを設けてもよい。その場合は、終了処理（ステップS514）において、そのアプリケーションが使用したクラスのカウンタを減じる。そして、カウンタが0になった場合は、もうそのクラスを使用しているアプリケーションは無いということで、クラスをアンロードできる。こうして、複数のアプリケーションが使用していたクラスでもアンロードできる。

【0091】また、上記の例では、バイトコードをロードした共用領域202はRAM上にあることを前提としていたが、ROM上に置いてもよい。バイトコードはアプリケーションの動作状況に関わらず、不変であるからである。その場合は、ロード、アンロードのステップは必要なくなり、さらに、クラス検索パスデータと検索ステップも不要になる。

【0092】以上、本発明について、上記の実施形態に基づき説明してきた。本発明はハードウエアを用いても、ソフトウエアでも実現できる。

【0093】従って、上述した実施形態の機能を実現するソフトウエアのプログラムコードを記憶した記憶媒体も、それを読み出して実行すれば上述の機能を実現でき、本発明を構成するものである。

【0094】

【発明の効果】以上説明したように、本発明によれば、インタプリタで動作させる機能を全て、事前に1個のアプリケーションとして構成することなく、複数のアプリケーションを同時に実行することができるようになり、また、多機能を簡単に実現でき、更には、プログラム記述の柔軟性、拡張性を増すことができるといった効果がある。

【0095】また、その際、インタプリタのインスタンスは1つで済むため、メモリの節約ができるといった効果がある。

【0096】また、複数アプリケーションを選択、実行させるための選択実行プログラムが不要となり、余計な手間が省けるといった効果がある。

【0097】また、クラス変数の使用法に制限を加えず、複数アプリケーションの正確な、予期したとおりの実行が可能なので、プログラミングの際、余計な制限事項を気にする必要がなくなり、また、プログラミングしやすくり、かつ、既存のアプリケーションも正しく動作させることができるといった効果がある。

【図面の簡単な説明】

【図1】本実施形態におけるインタプリタシステムのブロック図である。

【図2】インタプリタ内部のメモリ内容を示した図である。

【図3】クラス変数テーブルの構造を示した図である。

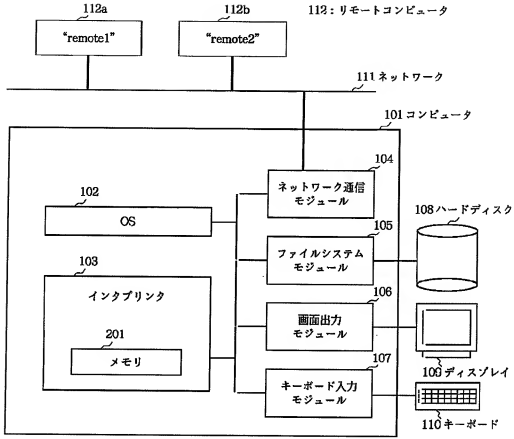
【図4】インタプリタの動作手順を示したフローチャートである。

【図5】アプリケーション実行時の動作手順を示したフローチャートである。

【符号の説明】

101 コンピュータ
102 OS
103 インタプリタ
104 ネットワーク通信モジュール
105 ファイルシステムモジュール
106 画面出力モジュール
107 キーボード入力モジュール
108 ハードディスク
109 ディスプレイ
110 キーボード
111 ネットワーク
112 リモートコンピュータ
201 メモリ
202 共用領域
203 アプリ個別領域
301 クラス変数テーブル

【図1】



【図3】

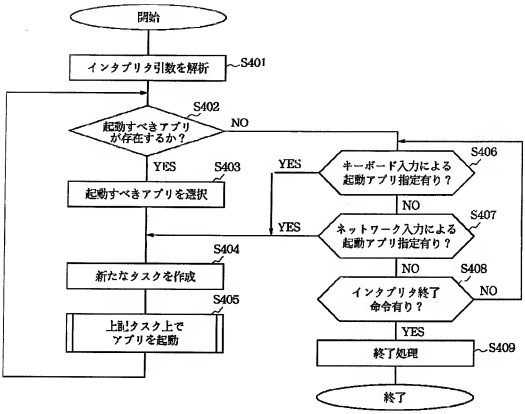
301 クラス変数テーブル

クラス		X			Y		Z		...	
クラス変数		x	y	z	...	...	...	...	...	...
クラス	A	301a	301b	301c						
	B	301d	301e	301f						
	C	301g	301h	301i						

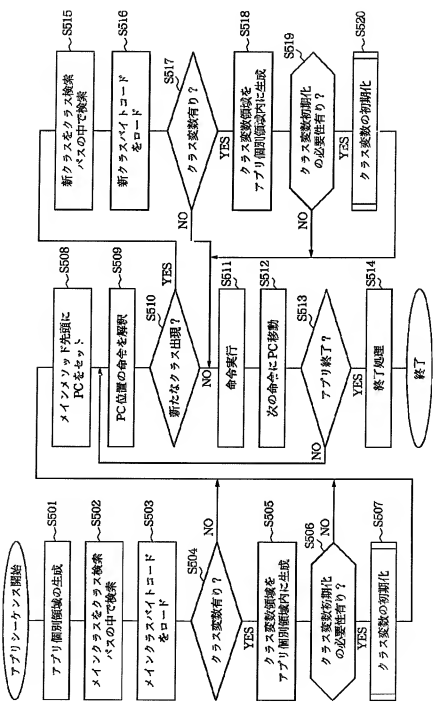
【図2】

201 メモリ
202 共用領域
クラスX バイトコード
クラスY バイトコード
クラスZ バイトコード
…
その他
203a アプリ個別領域（アプリA）
クラスX クラス変数領域
クラスY クラス変数領域
クラスZ クラス変数領域
…
各種オブジェクトデータ領域
スタック
その他
203b アプリ個別領域（アプリB）
クラスX クラス変数領域
クラスY クラス変数領域
クラスZ クラス変数領域
…
各種オブジェクトデータ領域
スタック
その他
203c アプリ個別領域（アプリC）
クラスX クラス変数領域
クラスY クラス変数領域
クラスZ クラス変数領域
…
各種オブジェクトデータ領域
スタック
その他

【図4】


開始
インタプリタ引数を解析
S401
S402
起動すべきアプリが存在するか？
NO
YES
S403
起動すべきアプリを選択
S404
新たなタスクを作成
S405
上記タスク上でアプリを起動
YES
S406
キーボード入力による起動アプリ指定有り？
NO
S407
ネットワーク入力による起動アプリ指定有り？
NO
S408
インタプリタ終了命令有り？
NO
YES
終了処理
S409
終了

【図5】


アプリシーケンス開始
S501
アプリ属性領域の生成
S502
メインクラスをクラス検索パスの中で検索
S503
メインクラスバイトコードをロード
S504
クラス変数有り？
YES
NO
S505
クラス変数領域をアプリ属性領域内に生成
S506
クラス変数初期化の必要性有り？
YES
NO
S507
クラス変数の初期化
S508
メインメソッド先頭にPCをセット
S509
PC位置の命令を解釈
S510
新たなクラス出現？
YES
NO
S511
命令実行
S512
次の命令にPC移動
S513
アプリ終了？
NO
YES
S514
終了処理
終了
S515
新クラスをクラス検索パスの中で検索
S516
新クラスバイトコードをロード
S517
クラス変数有り？
YES
NO
S518
クラス変数領域をアプリ属性領域内に生成
S519
クラス変数初期化の必要性有り？
YES
NO
S520
クラス変数の初期化